Polynomial Time Fragments of XPath with Variables

Emmanuel Filiot, Joachim Niehren, Jean-Marc Talbot, Sophie Tison

Polynomial Time Fragments of XPath with Variables

Emmanuel Filiot[1] Joachim Niehren[1] Jean-Marc Talbot[2] Sophie Tison[3]
[1] INRIA Futurs, Lille, Mostrare Project, [2] University of Provence, LIF, Marseille
[3] University of Lille 1 (LIFL, UMR 8022 of CNRS), Mostrare Project

ABSTRACT

Variables are the distinguishing new feature of XPath 2.0 which permits to select n-tuples of nodes in trees. It is known that the Core of XPath 2.0 captures n-ary first-order (FO) queries modulo linear time transformations. In this paper, we distinguish a fragment of Core XPath 2.0 that remains FO-complete with respect to n-ary queries while enjoying polynomial-time query answering.

Categories and Subject Descriptors: H.2.3 [*Languages*]: Query Languages

General Terms: Algorithms, Languages, Theory

Keywords: XML, XPath, Logic, N-ary queries

1. INTRODUCTION

XPath 2.0 is a common sublanguage of XQuery 1.0 and XSLT 2.0 which is recommended by the W3C. XPath 2.0 is a navigational language for selecting n-tuples of nodes in trees, which in contrast to its predecessors is designed to capture the expressiveness of first-order (FO) logic. Variables are the distinguishing new feature of XPath 2.0 over 1.0 beside of intersection and complementation.

Variables always existed in XQuery but were pushed down to XPath only recently. Much of what could only be done in XQuery before can now be done in XPath 2.0. For illustration, consider the following program of XQuery that extracts author-title pairs in books of a bibliography:

```
for $x in doc(''bib.xml'')/descendant::book return
  for $y in $x/child::author return
    for $z in $x/child::title return
      <pair> {$y} {$z} </pair>
```

In XPath 2.0, the same query can be expressed differently in a less frequent style. We use a pair of free variables ($y,$z) in order to select pairs of author-title nodes on the following navigation path:

```
doc(''bib.xml'')/descendant::book
    [    child::author[. is $y]
     and child::title [. is $z] ]
```

This example illustrates two of the three roles of variables in XPath 2.0. First of all, they allow to define n-tuples of nodes in trees by selecting intermediate nodes on paths. Second, they appear in node tests for expressing node equality. And third, they are needed in universal and existential quantifiers of XPath 2.0.

Practical path languages proposed by the W3C have large descriptions. Therefore, it has become good practice to study such languages within small formal cores which ignore syntactical details. More essentially, they abstract from standard operations on data values such as arithmetic's, which quickly lead to undecidability.

Gottlob, Koch, and Pichler [13] define *Core XPath* or synonymously *Core XPath 1.0*, the navigational core of XPath 1.0. They present an efficient algorithm for answering monadic queries in combined linear time. Marx analyzes the expressiveness [16, 15]. He shows that closure of *Core XPath* under complementation can express all binary FO queries. Much further work has been done on *Core XPath 1.0*. See [5] for an overview.

Ten Cate and Marx [22] distinguish the counterpart *Core XPath 2.0*, and show that query equivalence is decidable via a complete axiomatization. Since quantifiers are provided as primitives, it is not difficult to see that *Core XPath 2.0* captures the class of all n-ary FO queries modulo linear time transformations. This implies PSPACE completeness of model checking for *Core XPath 2.0*, so that one cannot hope for polynomial time query answering except if P=PSPACE.

In this paper, we distinguish a polynomial time fragment of *Core XPath 2.0* that we call the *polynomial time path language (*PPL*)*. We impose three main restrictions: no quantifiers, no variable sharing in path composition, and no variables below complementation. Note that all these conditions hold for the above example.

Our main result is that PPL still captures the class of all n-ary FO queries, while allowing for polynomial time query answering without any bound on n in the size of the answer set, the query, the tree t, and tuple width n. Note that the size of the answer set can grow up to the overall number of n-tuples $|t|^n$, even though it is often much smaller. In practice, n can easily get up to 10 or more, for instance, when querying for attributes of restaurants such as name, address, phone number, fax number, street, street number, district, city, country, average menu price, food style, etc. Thus, the time for query answering should be polynomial in the size of the answer set, rather than in the number of n-tuples.

We start from the binary query language PPL^{bin} which extracts the essence of PPL without variables. This language can be identified with the extension of *Core XPath 1.0* by complementation. Due to Marx's result stated above, PPL^{bin} can define all binary FO queries, i.e. PPL^{bin} = FO^{bin} (modulo non-elementary transformations). Simultaneously, PPL^{bin} enjoys polynomial time query answering for binary queries. This requires an extension of the efficient algorithm for *Core XPath 1.0* in order to deal with more

PODS'07, June 11–14, 2007, Beijing, China.

costly intersection and complementation operators of *Core XPath 2.0*.

In the second step, we propose the *hybrid composition language* $\mathrm{HCL}(L)$ for defining path languages with variables systematically from binary query languages L. Path expressions $C \in \mathrm{HCL}(L)$ are either binary queries $q \in L$, variables x, compositions C/C', unions $C \cup C'$, or tests $[C]$. Neither complementation nor transitive closure are provided. Conjunctive and disjunctive queries over L are expressible. Indeed, the language $\mathrm{HCL}(L)$ can be encoded into hybrid logic with modalities in L [6], i.e. modal logics for unranked trees with variables [4, 2].

We are mostly interested in the fragment of $\mathrm{HCL}(L)$ where variable sharing is forbidden in path compositions. This fragment that we call $\mathrm{HCL}^-(L)$ subsumes unions of acyclic conjunctive queries (ACQ) over L [24]. It is more general in that unions are permitted in arbitrary positions, not only on top level of ACQs. This extension does not increase expressiveness though. We present a new algorithm for answering n-queries in $\mathrm{HCL}^-(L)$ in polynomial time that uses a query answering algorithm for L as an oracle (which always responds in constant time).

The polynomial time algorithm for PPL is induced by the polynomial time algorithm for PPL^{bin} via the following language equivalence, which we prove modulo linear time transformations:

$$\mathrm{HCL}^-(\mathrm{PPL}^{bin}) = \mathrm{PPL}$$

The same equivalence can be used to prove the completeness of PPL with respect to n-ary FO queries since $\mathrm{PPL}^{bin} = \mathrm{FO}^{bin}$ as shown by Marx. It remains to show that hybrid compositions of binary FO queries without variable sharing capture n-ary FO-queries:

$$\mathrm{HCL}^-(\mathrm{FO}^{bin}) = \mathrm{FO}$$

This follows by combining two results of Marx in [15] (one of which strengthens the result of [19]). These show that unions of ACQs over FO^{bin} can express all n-ary FO queries. Even though stated only for binary trees it carries over to unranked trees as we show. Finally recall that $\mathrm{HCL}^-(\mathrm{FO}^{bin})$ subsumes unions of ACQs over FO^{bin}.

Plan of the Paper.. We recall *Core XPath 2.0* in Section 2 and distinguish its polynomial time fragment PPL in Section 3. The variable-free part PPL^{bin} is discussed in Section 4. The hybrid composition language $\mathrm{HCL}(L)$ is introduced in Section 5 and then applied to PPL^{bin}. The relation to acyclic conjunctive queries is discussed in Section 6. Polynomial time query answering for $\mathrm{HCL}^-(L)$ is discussed in Section 7. Section 8 presents a direct proof of $\mathrm{HCL}^-(\mathrm{FO}^{bin}) = \mathrm{FO}$. Related work on composition languages is discussed in Section 9.

2. CORE XPATH 2.0

The navigational core of XPath 2.0 (*Core XPath 2.0*) has been proposed very recently by ten Cate and Marx [22]. We recall their definition and known results on the relation with FO logic.

XPath 1.0 is a variable-free language for defining monadic or binary queries. Its path expressions can be understood as navigation plans, i.e. on how to navigate from a start node in a tree to a target node. They define monadic queries, when starting at the root, or binary queries relating start nodes to end nodes. All these queries are FO definable. XPath 2.0 is a language for defining n-ary queries. The most important extensions with respect to XPath 1.0 are variables, complement and intersection on path level, and FO quantifiers.

The study of expressiveness and efficiency issues for XPath are usually based on small formal core languages. These extract the navigational aspects of XPath rather than operation on data values or arithmetics. The data model of XPath is abstracted to unranked sibling-ordered trees with nodes labeled in some set Σ. Other features are ignore, such as attributes, data values, and name spaces. Such restrictions are unavoidable since decidability quickly fails otherwise.

An unranked tree $t \in T_\Sigma$ is a pair $a(t_1 \dots t_n)$ consisting of a label $a \in \Sigma$ and a possibly empty sequence of trees $t_1 \dots t_n \in T_\Sigma$. Every tree t defines a logical structure, whose domain is $\mathtt{nodes}(t)$, the set of nodes of t. Its signature contains all axis of XPath as well as their transitive closure. Furthermore there are monadic predicates lab_a for all labels $a \in \Sigma$. We use shortcuts $\mathsf{ch} = \mathsf{child}$ and $\mathsf{ns} = \mathsf{nextsibling}$. The interpretation $p(t) \subseteq \mathtt{nodes}(t)^n$ is the set of n-tuples that a predicate p of arity n selects in tree t.

As usual in XPath related research [14, 15], we study the FO logic over unranked trees with predicates ch^* and ns^*. Let $x, y, z \in \mathtt{Var}$ range over variables. We will work with FO formulas ϕ with the following abstract syntax.

$$\phi := \mathsf{ns}^*(x,y) \mid \mathsf{ch}^*(x,y) \mid \mathsf{lab}_a(x) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x \phi$$

Judgments $t, \alpha \models \phi$ state that tree $t \in T_\Sigma$ is a model of ϕ under variable assignment $\alpha : \mathtt{Var} \to \mathtt{nodes}(t)$. These judgments are defined in the usual Tarskian manner. Note that all axis of XPath are definable in FO as well as their transitive closures, even though we use only a restricted set in the formulas. Node equality is definable too.

An *n-ary query* is a function mapping trees $t \in T_\Sigma$ to subsets of $\mathtt{nodes}(t)^n$. Pairs of FO formulas ϕ and sequences of n variables $\overline{x} = x_1 \dots x_n$ define n-ary queries $q_{\phi,\overline{x}}$ which satisfy for all trees $t \in T_\Sigma$:

$$q_{\phi,\overline{x}}(t) = \{(\alpha(x_1), \dots, \alpha(x_n)) \mid t, \alpha \models \phi\}$$

Let FO be the language of all n-ary FO queries and FO^{bin} the language of all binary FO queries.

The syntax of *Core XPath 2.0* is listed in Fig. 1. *Core XPath 2.0* extends the syntax of *Core XPath 1.0* by node variables $\$x$, quantified path expression for $\$x$ in P_1 return P_2, relative complements P_1 except P_2 and intersections P_1 intersect P_2. The size of terms $|P|$, $|T|$, and $|\phi|$ is the number of nodes in these expressions. We write $\mathtt{Var}(P)$, $\mathtt{Var}(T)$, and $\mathtt{Var}(\phi)$ for the set of free variables in the respective terms.

The semantics of *Core XPath 2.0* is given in Fig. 2. Path expressions P denote sets of node pairs $[\![P]\!]^{t,\alpha} \subseteq \mathtt{nodes}(t)^2$ for all trees t and variable assignments $\alpha : \mathtt{Var} \to \mathtt{nodes}(t)$. Test expressions T denote sets of nodes $[\![T]\!]^{t,\alpha}_{\text{test}} \subseteq \mathtt{nodes}(t)$. A variable path expression $\$x$ requires to move from the current node to the node pointed by $\$x$. This value can be constrained further by node equality constraints in test expressions $[\$x \text{ is } .\,]$ or $[\$x \text{ is } \$y]$.

We call two path expressions P_1 and P_2 equivalent if $[\![P_1]\!]^{t,\alpha} = [\![P_2]\!]^{t,\alpha}$ for all trees t and variable assignments α into nodes of t.

XPath 2.0 has been designed to feature the expressiveness of FO. To see this, we start with an auxiliary path expression by which to reach all nodes in a tree:

```
nodes =
   (ancestor::* union .) / (descendant::* union .)
```

Now we translate FO formulas as follows into *Core XPath 2.0*.

$(\!|\exists x.\phi|\!) = \text{for } \$x \text{ in nodes return } (\!|\phi|\!)$
$(\!|\neg\phi|\!) = .[\text{not } (\!|\phi|\!)]$
$(\!|\phi \wedge \phi'|\!) = (\!|\phi|\!)/(\!|\phi'|\!)$
$(\!|\mathsf{ns}^*(x,y)|\!) = \$x/(\text{following\_sibling :: * union .})/[.\text{ is } \$y]$
$(\!|\mathsf{ch}^*(x,y)|\!) = \$x/(\text{descendant :: * union .})/[.\text{ is } \$y]$

```
Axis      := self | child | parent
           | descendant | ancestor
           | following_sibling
           | preceding_sibling

NameTest  := QName | *    where QName ∈ Σ
Step      := Axis::NameTest
NodeRef   := . | $x       where x ∈ Var

PathExpr  := Step | NodeRef
           | PathExpr/PathExpr
           | PathExpr union PathExpr
           | PathExpr intersect PathExpr
           | PathExpr except PathExpr
           | PathExpr[TestExpr]
           | for $x in PathExpr return PathExpr

TestExpr  := PathExpr | CompTest | not TestExpr
           | TestExpr and TestExpr
           | TestExpr or TestExpr

CompTest  := NodeRef is NodeRef
```

Figure 1: Syntax of Core XPath 2.0

$$[\![A::N]\!]^{t,\alpha} = \{(v_1, v_2) \in A(t) \mid v_2 \in \mathsf{lab}_N(t)\}$$
$$[\![A::*]\!]^{t,\alpha} = A(t)$$
$$[\![.]\!]^{t,\alpha} = \{(v, v) \mid v \in \mathtt{nodes}(t)\}$$
$$[\![\$x]\!]^{t,\alpha} = \{(v, \alpha(x)) \mid v \in \mathtt{nodes}(t)\}$$

$$[\![P_1/P_2]\!]^{t,\alpha} = [\![P_1]\!]^{t,\alpha} \circ [\![P_2]\!]^{t,\alpha}$$
$$[\![P_1 \text{ union } P_2]\!]^{t,\alpha} = [\![P_1]\!]^{t,\alpha} \cup [\![P_2]\!]^{t,\alpha}$$
$$[\![P_1 \text{ intersect } P_2]\!]^{t,\alpha} = [\![P_1]\!]^{t,\alpha} \cap [\![P_2]\!]^{t,\alpha}$$
$$[\![P_1 \text{ except } P_2]\!]^{t,\alpha} = [\![P_1]\!]^{t,\alpha} - [\![P_2]\!]^{t,\alpha}$$
$$[\![P[T]]\!]^{t,\alpha} = \{(v_1, v_2) \in [\![P]\!]^{t,\alpha} \mid v_2 \in [\![T]\!]^{t,\alpha}_{\text{test}}\}$$
$$[\![\text{for } \$x \text{ in } P_1 \text{ return } P_2]\!]^{t,\alpha} = \{(v_1, v_3) \mid \exists v_2 \in \mathtt{nodes}(t). (v_1, v_2) \in [\![P_1]\!]^{t,\alpha} \text{ and } (v_1, v_3) \in [\![P_2]\!]^{t,\alpha[x \mapsto v_2]}\}$$
$$[\![P]\!]^{t,\alpha}_{\text{test}} = \{v \mid (v, v') \in [\![P]\!]^{t,\alpha}\}$$
$$[\![. \text{ is } \$x]\!]^{t,\alpha}_{\text{test}} = \{\alpha(x)\}$$
$$[\![. \text{ is } .]\!]^{t,\alpha}_{\text{test}} = \mathtt{nodes}(t)$$
$$[\![\$x \text{ is } \$y]\!]^{t,\alpha}_{\text{test}} = \{\alpha(x) \mid \alpha(x) = \alpha(y)\}$$
$$[\![\text{not } T]\!]^{t,\alpha}_{\text{test}} = \mathtt{nodes}(t) - [\![T]\!]^{t,\alpha}_{\text{test}}$$
$$[\![T_1 \text{ and } T_2]\!]^{t,\alpha}_{\text{test}} = [\![T_1]\!]^{t,\alpha}_{\text{test}} \cap [\![T_2]\!]^{t,\alpha}_{\text{test}}$$
$$[\![T_1 \text{ or } T_2]\!]^{t,\alpha}_{\text{test}} = [\![T_1]\!]^{t,\alpha}_{\text{test}} \cup [\![T_2]\!]^{t,\alpha}_{\text{test}}$$

Figure 2: Semantics of Core XPath 2.0

This translation doesn't care much about the current position during navigation. At each time, it simply jumps to the node denoted by some variable and tests some literal there. The encoding is correct in following sense.

LEMMA 1. *For all trees t and assignments α from* $\mathtt{Var}$ *into* $\mathtt{nodes}(t)$*:*

$$t, \alpha \models \phi \textit{ iff } [\![(\!|\phi|\!)]\!]^{t,\alpha} \neq \emptyset$$

PROOF. By induction on the structure of FO-formulas. We only elaborate the most interesting case, which is that of quantification. The following statements are equivalent: $t, \alpha \models \exists x.\phi$ iff exists $v \in \mathtt{nodes}(t)$ s.t. $t, \alpha[x \mapsto v] \models \phi$ iff $\cup_{v \in \mathtt{nodes}(t)} [\![(\!|\phi|\!)]\!]^{t,\alpha[x \mapsto v]} \neq \emptyset$ by induction hypothesis iff $[\![\text{for } \$x \text{ in nodes return } (\!|\phi|\!)]\!]^{t,\alpha} \neq \emptyset$ iff $[\![(\!|\exists x \phi|\!)]\!]^{t,\alpha} \neq \emptyset$. □

This translation maps conjunction to path composition. Alternatively, one could use the intersect operator, or conjunctions in tests. Negation is mapped to negation in tests but can equally be expressed by the except operator. Thus, there is some redundancy in the language. In particular, we can remove the intersect operator by using the following equivalence:

```
P1 intersect P2 = P1 except (nodes except P2)
```

N-ary queries in *Core XPath 2.0* can be specified by a path expression P and a sequence of variables $\overline{x} = x_1, \ldots, x_n$. The query $q_{P,\overline{x}}$ that they define satisfies for all trees t that:

$$q_{P,\overline{x}}(t) = \{(\alpha(x_1), \ldots, \alpha(x_n)) \mid [\![P]\!]^{t,\alpha} \neq \emptyset\}$$

Neither the start node of navigation nor the end node play any particular role here, since these can always be accessed by variables. Furthermore, we can fix the start point $\$x$ of navigation along some path P to be the root by prefixing this equality constraint to P:

$$.[. \text{ is } \$x \text{ and not}(\text{parent} :: *)]/P$$

PROPOSITION 1. *Core XPath 2.0 and FO capture the same n-ary queries modulo linear time transformations.*

$$\textit{Core XPath 2.0} = \text{FO}$$

PROOF. The above transformation of FO to *Core XPath 2.0* is in linear time and preserves queries by Lemma 1. Conversely, it is quite obvious that all n-ary queries expressible in *Core XPath 2.0* are FO definable. It is sufficient to introduce existentially quantified variables systematically for all positions in path expressions. Note that existential quantifiers can not be dropped when below negation. Note also that all axis of *Core XPath 2.0* are definable in FO, so that the back transformation is in linear time too. □

Model checking for FO is PSPACE complete [21]. This is the problem of answering Boolean queries, i.e. queries of arity 0. By Proposition 1, this result carries over to *Core XPath 2.0*.

COROLLARY 1. *Model checking for Core XPath 2.0 is PSPACE-complete.*

3. POLYNOMIAL-TIME PATH LANGUAGE

We present restrictions on *Core XPath 2.0* in order to define a large polynomial-time fragment that we call the *polynomial-time path language (*PPL*)*.

First of all, we disallow for-loops in order to avoid quantifier alternation, which clearly raises PSPACE-hardness of model checking.

LEMMA 2. *Quantifier free F0 formulas can be translated in linear time into queries by formulas of Core XPath 2.0 without* for-loops*.*

PROOF. The translation $(\!|\phi|\!)$ maps quantifier free FO-formulas to paths expressions without for-loops. Conversely, the introduction of variables for all positions in a path expression P does not require any quantifier. □

The converse of Lemma 2 is false. For instance, the following negative formula expresses that if x is an ancestor of y, no single ns step occurs on the path from x to y:

$$\neg(\exists z.\exists z'.(\mathsf{ch}^*(x, z) \wedge \mathsf{ns}(z, z') \wedge \mathsf{ch}^*(z', y)))$$

We could not eliminate quantifiers in this formula. Nevertheless, it can be expressed in *Core XPath 2.0* without for-loops:

$$.[\text{not } (\$x/\text{descendant}::*/\text{nextsibling}::*/\text{descendant}::*[. \text{ is } \$y])]$$

The translation $(\!| \, . \, |\!)$ of this path expression to an FO formula creates existentially quantified variables below negation. This quantifier cannot be dropped or eliminated, even if we add firstchild to the signature.

To see this, let $\phi_0(x, y)$ be the counter example above and suppose that it can be expressed by some existential formula of the form $\exists x_1 \ldots \exists x_n . \xi$ where ξ is a quantifier free FO formula with free variables $x_1, \ldots, x_n, x, y$.

Let t be a tree with nodes u, v satisfying $(u, v) \in \mathsf{firstchild}(t)^{n+2}$. This means that one can descend from u to v over a path of $n + 2$ subsequent firstchild steps. Since $t, u, v \models \phi_0(x, y)$ there exists a variable assignment α with $\alpha(x) = u, \alpha(y) = v$ such that $t, \alpha \models \xi$. Since the number of intermediate nodes on the path from u to v is $n + 1$, there exists a least one node w on this path that does not belong to $\alpha(\{x_1, \ldots, x_n, x, y\})$. Let us transform the tree t into a tree t' by adding a new node on the left of the previous first child of w. One can prove that $t', \alpha \models \xi$, since on the nodes of t we only changed the relation firstchild for w. This contradicts $t', u, v \not\models \phi_0$ which fails since the path from u to v contains a nextsibling step now.

PROPOSITION 2 (QUANTIFIER ELIMINATION).
Core XPath 2.0 without for *loops captures the set of all n-ary FO-definable queries, modulo a non-elementary transformation.*

PROOF. For the case of binary queries this result has been shown by Marx. More generally, all extensions of Core XPath that are closed under complementation are complete with respect to binary FO-queries (Theorem 2 of [16]). In Corollary 4.2 of [15], Marx shows that n-ary FO queries can be expressed as union of acyclic conjunctive queries over binary FO queries. It concludes the proof, since ACQs over binary *Core XPath* queries with except can easily be expressed in *Core XPath 2.0* (Proposition 4). □

Removing for-loops, and even variables below negations, from *Core XPath 2.0*, however, does not suffice to reduce the complexity of query non-emptiness.

PROPOSITION 3. *Query non-emptiness for Core XPath 2.0 without* for*-loops and variables below negation is NP-complete.*

One can prove this result by reduction from Sat. The encoding of Sat relies on using variable sharing between different branches of compositions. Similar effects can arise with filters and conjunctions in tests, so we have to forbid variable sharing in all these cases.

DEFINITION 1. *The* polynomial-time path language (PPL) *is the restriction of Core XPath 2.0 whose expressions satisfy the following conditions:*

N(for) *no* for *loops and thus no explicit quantifiers.*

NV(intersect) *no variables in intersections: all subexpressions* P_1 *intersect* P_2 *satisfy* $\mathtt{Var}(P_1) = \mathtt{Var}(P_2) = \emptyset$.

NV(except) *no variables in exceptions: all subexpressions* P_1 *except* P_2 *satisfy* $\mathtt{Var}(P_1) = \mathtt{Var}(P_2) = \emptyset$.

NV(not) *no variables below negation, i.e., subexpressions* not(T) *satisfy* $\mathtt{Var}(T) = \emptyset$

NVS(/) *no variable sharing in composition: all subexpressions* P_1/P_2 *satisfy* $\mathtt{Var}(P_1) \cap \mathtt{Var}(P_2) = \emptyset$

NVS([]) *no variable sharing in filters: all subexpressions* $P[T]$ *satisfy* $\mathtt{Var}(P) \cap \mathtt{Var}(T) = \emptyset$

```
PathExpr  := Axis::NameTest
           | PathExpr/PathExpr
           | PathExpr union PathExpr
           | except PathExpr
           | [PathExpr]
```

Figure 3: Syntax of PPL^{bin}

$(\!| . |\!) = \mathsf{self}$
$(\!| P \text{ intersect } P' |\!) = \mathsf{except}\ (\mathsf{except}\ (\!| P |\!)\ \mathsf{union}\ \mathsf{except} (\!| P' |\!))$
$(\!| P \text{ except } P' |\!) = \mathsf{except}\ (\mathsf{except}\ (\!| P |\!)\ \mathsf{union}\ (\!| P' |\!))$
$(\!| P[T] |\!) = (\!| P |\!)/(\!| [T] |\!)_{\text{test}}$
$(\!| [T \text{ and } T'] |\!)_{\text{test}} = (\!| [T] |\!)_{\text{test}}/(\!| [T'] |\!)_{\text{test}}$
$(\!| [T \text{ or } T'] |\!)_{\text{test}} = (\!| [T] |\!)_{\text{test}}\ \mathsf{union}\ (\!| [T'] |\!)$
$(\!| [\text{not } P] |\!)_{\text{test}} = [\mathsf{except}\ (\!| P |\!)]$
$(\!| [\text{not } (T \text{ and } T')] |\!)_{\text{test}} = (\!| [\text{not } T] |\!)_{\text{test}}\ \mathsf{or}\ (\!| [\text{not } T'] |\!)_{\text{test}}$
$(\!| [\text{not } (T \text{ or } T')] |\!)_{\text{test}} = (\!| [\text{not } T] |\!)_{\text{test}}\ \mathsf{and}\ (\!| [\text{not } T'] |\!)_{\text{test}}$
$(\!| [\text{not not } T] |\!)_{\text{test}} = (\!| [T] |\!)_{\text{test}}$
$(\!| [. \text{ is } .] |\!)_{\text{test}} = \mathsf{self}$

Figure 4: From *Core XPath 2.0* $\cap$ N($\$x$) to PPL^{bin}

NVS(and) *no variable sharing in conjunctions: all tests in subexpressions* T_1 *and* T_2 *satisfy* $\mathtt{Var}(T_1) \cap \mathtt{Var}(T_2) = \emptyset$

Variables on the right of exceptions are forbidden, since this is a form of negation. They are forbidden on the left too, since this is a form of intersection. Variables sharing in filters would amount to variable sharing in composition.

There are no restrictions on the union and or operators, so variables can be shared there. It is crucial to forbid variables below all kinds of negation in except and not . Otherwise, one could encode intersect and and expressions with variable sharing.

THEOREM 1 (POLYNOMIAL-TIME PATH LANGUAGE).
Expressiveness. PPL *can express all n-ary first-order queries.*

Complexity. *Answers sets of n-ary queries* $A = q_{P,\overline{x}}$ *for expression* P *of* PPL *can be answered in polynomial time* $O(|P||t|^3 + n\,|P|\,|t|^2\,|A|)$ *where* $|A|$ *is the cardinality of the answer set.*

The expressiveness statement will be proved in Corollary 2, the efficiency part is the combination of Corollary 3, Proposition 5 and 2.

4. THE VARIABLE-FREE FRAGMENT

We present a minimalistic dialect of PPL without variables that we call the *binary polynomial-time path language (*PPL^{bin} *)*. This language can be identified with the variable-free fragment of *Core XPath 2.0* and with the extension of *Core XPath 1.0* by the except operator.

The syntax of PPL^{bin} is given in Fig. 3. It is a proper sublanguage of PPL with a minor exception that we have restricted the except operator to its negative side: except P = nodes except P. All expressions of PPL^{bin} are variable free:

N($\$x$) *there are no variables, no* for *loops, no node comparisons.*

The expressions of PPL^{bin} define binary queries – equally to path expressions of *Core XPath 2.0*– which relate start nodes of navigation to end nodes:

$$q_P^{bin} = q_{P',(x,y)} \qquad \text{where} \quad P' = [. \text{ is } \$x]/P/[. \text{ is } \$y]$$

PROPOSITION 4. *The following 4 languages define the same binary queries modulo linear time translations:*

$$\begin{array}{rcl} \text{PPL}^{bin} & = & \text{PPL} \cap \text{N}(\$x) \\ & = & \textit{Core XPath 1.0} \cup \textit{except} \\ & = & \textit{Core XPath 2.0} \cap \text{N}(\$x) \end{array}$$

PROOF. The inclusions from the left to the right are all obvious by syntactic identity. For the converse, we encode *Core XPath 2.0*$\cap$ N($\$x$) into PPL^{bin} . This is done by the linear time translation in Fig. 4. □

THEOREM 2. PPL^{bin} *enjoys the following two properties:*

Expressiveness. *All binary FO-queries can be expressed in* PPL^{bin} .

Complexity *Query answering for expression P of* PPL^{bin} *which outputs the set of node pairs for a given tree t is in time $O(|P||t|^3)$.*

The completeness result with respect to FO expressiveness has been shown by Marx [16]. It applies more generally to all extensions of *Core XPath 1.0* that are closed under complementation. Another instance beside *Core XPath* with except is *Conditional XPath*.

For the algorithmic part, it is well known that unary queries in *Core XPath 1.0* can be answered in linear time [13]. This gives a quadratic binary query answering algorithm for *Core XPath 1.0*, in the size of the input tree. The main evaluation trick of *Core XPath 1.0* lies in that the set of successors $S_a(N) = \{u' \mid \exists u \in N,\ a(u,u')\}$ of a set of nodes N by a standard axis a, in a tree t, is computable in linear time $O(|t|)$. This can be extended to full *Core XPath 1.0* expression, so that computing $S_P(N)$, for some *Core XPath 1.0* expression P, is in linear time $O(|P||t|)$. However, it is not clear whether this trick can be used for evaluating PPL^{bin} , since except operator can occur at any position in the input expression, and in general $S_{\mathsf{except}\ P}(N) \neq \overline{S_P(N)}$.

We now give an algorithm to answer binary queries by PPL^{bin} expressions. Given a PPL^{bin} expression P, and a tree t, we represent $q_P^{bin}(t)$ as a $|t| \times |t|$ Boolean matrix M_P^t. Obviously, M_P^t is defined by $\forall u, u' \in \texttt{nodes}(t)$, $M_P^t[u,u'] = 1$ iff $(u,u') \in q_P^{bin}(t)$. We consider the following operations on matrix: the sum $+$ and the product . over the Boolean algebra $(\{0,1\}, \vee, \wedge)$. We also defined $\neg M$ as the matrix M where 0's (resp. 1's) have been replaced by 1's (resp. 0's), and $[M]$ by $\forall u, u' \in \texttt{nodes}(t)$ $[M][u,u'] = 1$ iff $u = u'$ and $\exists u'' \in \texttt{nodes}(t)$ s.t. $M[u,u''] = 1$. We get the following, for all PPL^{bin} expressions P_1, P_2, P and all trees t:

$$\begin{array}{rclrcl} M^t_{P_1/P_2} & = & M^t_{P_1}.M^t_{P_2} & M^t_{\mathsf{except}P} & = & \neg M^t_P \\ M^t_{P_1 \,\mathsf{union}\, P_2} & = & M^t_{P_1} + M^t_{P_2} & M^t_{[P]} & = & [M^t_P] \end{array}$$

A naive implementation of these operations leads to a $O(|P||t|^3)$ time complexity to evaluate a PPL^{bin} expression P on a tree t. However, from a theoretical point of view, this can be improved to $O(|P||t|^{2.376})$ since the best known algorithm for n-square Boolean matrix multiplication is in time $O(n^{2.376})$, due to Coppersmith and Winograd [7].

5. HYBRID COMPOSITION LANGUAGE

We define the hybrid composition language HCL(L) parametrized by some binary query language L.

In general, L consists of a set of expressions $b \in L$ that define binary queries q_b. In the simplest case, L could be the set of axis of *Core XPath 2.0*. Alternatively, it could be FO^{bin} or PPL^{bin} . In these cases, the sizes of the expressions of L are relevant to the complexity of query answering.

The hybrid composition language is navigational and provides variables similarly to *Core XPath 2.0*. Its expressions are listed in Fig. 5. They are expressions $b \in L$ defining binary queries, compositions C/C', variables $x \in Var$, filters $[C]$ or disjunctions $C \cup C'$. The semantics of these terms is given in Fig. 6. Expressions $b \in L$ are mapped to the binary queries q_b they define.

The only new part is the concept of variables in HCL(L). It is a little simpler than that of *Core XPath 2.0*; it is motivated by hybrid logic [6]. Variables x of HCL(L) are interpreted as node tests rather than goto commands $\$x$. Both of these concepts of variables are closely related:

$$\begin{array}{ll} \text{equality tests:} & x = [.\ \mathsf{is}\ \$x] \\ \text{goto commands:} & \$x = \mathsf{nodes}/x \end{array}$$

Note that this translation only works if the binary query nodes can be expressed in HCL(L). This is the case, for instance, if L contains ch* and root. Note also that complementation is not provided by HCL(L).

The *composition size* $|C|$ is the number of nodes of C. Note that expression for binary queries $b \in L$ have composition size 1, while their size as an expression in L may be different. We write $\texttt{Sub}(C)$ for the set of its subformulas.

Expressions C of HCL(L) with a sequence of n variables $\overline{x} = x_1 \ldots x_n$ define n-ary queries as before:

$$q_{C,\overline{x}}(t) \quad = \quad \{(\alpha(x_1), \ldots, \alpha(x_n)) \mid [\![C]\!]^{t,\alpha} \neq \emptyset\}$$

As stated in the introduction, we are mostly interested in the fragment of HCL(L) without variable sharing in composition, i.e., with expressions that satisfy NVS(/). We call this fragment HCL$^-(L)$.

PROPOSITION 5. *The following two languages define the same n-ary queries modulo linear time translations:*

$$\text{HCL}^-(\text{PPL}^{bin}) \ = \ \text{PPL}$$

PROOF. The inclusion from the left to the right follows from the following translation:

$$\begin{array}{l} (\!|P|\!) = P \text{ where } P \text{ in PPL}^{bin} \\ (\!|C/C'|\!) = (\!|C|\!)/(\!|C'|\!) \\ (\!|x|\!) = .[\ .\ \mathsf{is}\ \$x] \\ (\!|[C]|\!) = .[(\!|C|\!)] \\ (\!|C \cup C'|\!) = (\!|C|\!)\ \mathsf{union}\ (\!|C'|\!) \end{array}$$

It remains to verify that the result of the translation belongs to PPL. This holds since PPL$^{bin} \subseteq$ PPL and since we assume NVS(/). The inverse inclusion follows by the inverse translation in Fig. 7. It can be seen by induction on the size of path expressions that the back translation always maps to HCL$^-$(PPL^{bin}). □

THEOREM 3. *Hybrid compositions of binary FO-queries without variable sharing capture n-ary FO-queries:*

$$\text{HCL}^-(\text{FO}^{bin}) = \text{FO}$$

In Proposition 9 of Section 6, we will see that HCL$^-$(FO ^{bin}) subsumes unions of acyclic conjunctive queries over FO^{bin}. These capture FO by Corollary 4.2 of Marx [15]. An alternative direct proof is given in Section 8.

COROLLARY 2. PPL *can express all n-ary FO queries.*

PROOF. Theorem 2 of [15] shows that PPL^{bin} = FO^{bin}. Proposition 5 yields HCL$^-$(PPL^{bin}) $\subseteq$ PPL and hence HCL$^-$(FO^{bin}) $\subseteq$ PPL, so that Theorem 3 yields FO $\subseteq$ PPL. The converse has been argued in Proposition 1. □

$$
\begin{array}{rcll}
C & := & b & b \in L, \textit{ expression for binary query} \\
 & | & C/C' & \textit{composition} \\
 & | & x & \textit{variable} \\
 & | & [C] & \textit{filter} \\
 & | & C \cup C' & \textit{disjunction}
\end{array}
$$

Figure 5: Syntax of HCL(L)

$$
\begin{array}{l}
[\![b]\!]^{t,\alpha} = q_b(t) \\
[\![C/C']\!]^{t,\alpha} = [\![C]\!]^{t,\alpha} \circ [\![C']\!]^{t,\alpha} \\
[\![x]\!]^{t,\alpha} = \{(\alpha(x), \alpha(x))\} \\
[\![[C]]\!]^{t,\alpha} = \{(u,u) \mid \exists u'.(u,u') \in [\![C]\!]^{t,\alpha}\} \\
[\![C \cup C']\!]^{t,\alpha} = [\![C]\!]^{t,\alpha} \cup [\![C']\!]^{t,\alpha}
\end{array}
$$

Figure 6: Semantics of HCL(L)

$$
\begin{array}{l}
(\!|A :: N|\!)^{-1} = A :: N \qquad \text{(in PPL}^{bin}\text{)} \\
(\!|.|\!)^{-1} = \mathsf{self} \qquad \text{(in PPL}^{bin}\text{)} \\
(\!|\$x|\!)^{-1} = \mathsf{nodes}/x \qquad \text{(in PPL}^{bin}\text{)} \\
(\!|P_1/P_2|\!)^{-1} = (\!|P_1|\!)^{-1}/(\!|P_2|\!)^{-1} \qquad (\text{NVS}(/) \text{ is preserved}) \\
(\!|P_1 \mathsf{\ union\ } P_2|\!)^{-1} = (\!|P_1|\!)^{-1} \cup (\!|P_2|\!)^{-1} \\
(\!|P_1 \mathsf{\ intersect\ } P_2|\!)^{-1} = P_1 \mathsf{\ intersect\ } P_2 \qquad \text{(in PPL}^{bin} \\
\quad \text{modulo linear time by NV(intersect) and Proposition 4)} \\
(\!|P_1 \mathsf{\ except\ } P_2|\!)^{-1} = P_1 \mathsf{\ except\ } P_2 \qquad \text{(in PPL}^{bin} \\
\quad \text{modulo linear time by NV(except) and Proposition 4)} \\
(\!|P_1[P_2]|\!)^{-1} = (\!|P_1|\!)^{-1}/[(\!|P_2|\!)^{-1}] \qquad (\text{NVS}([]) \text{ ensures NVS}(/)) \\
(\!|P[\mathsf{not\ } T]|\!)^{-1} = (\!|P|\!)^{-1}/.[\mathsf{not\ } T] \qquad (.[\mathsf{not\ } T] \text{ is in PPL}^{bin} \\
\quad \text{by NV(not) and Proposition 4 so the result satisfies NVS}(/)) \\
(\!|P[T_1 \mathsf{\ and\ } T_2]|\!)^{-1} = (\!|P|\!)^{-1}/(\!|./[T_1]|\!)^{-1}/(\!|./[T_2]|\!)^{-1} \\
\quad (\text{NVS(and) guarantees NVS}(/)) \\
(\!|P[T_1 \mathsf{\ or\ } T_2]|\!)^{-1} = P/((\!|./[T_1]|\!)^{-1} \mathsf{\ union\ } (\!|./[T_2]|\!)^{-1}) \\
\quad (\text{NVS}([]) \text{ maps to NVS}(/))
\end{array}
$$

Figure 7: From PPL to HCL$^-$(PPL^{bin})

6. ACYCLIC CONJUNCTIVE QUERIES

We relate the hybrid composition language HCL(L) to positive quantifier free FO-queries over L, and its fragment without variable sharing in compositions HCL$^-(L)$ to unions of acyclic queries over L. This yields results on the efficiency and expressiveness of HCL$^-$(FO^{bin}) that we elaborate.

Positive quantifier-free FO formulas over L have the following abstract syntax where $b \in L$:

$$\xi ::= b(x,y) \mid x{=}y \mid \xi \wedge \xi' \mid \xi \vee \xi'$$

PROPOSITION 6. *If ch^* belongs to L then* HCL(L) *can express the same n-ary queries as positive quantifier-free FO formulas over L modulo linear time translations.*

PROOF. Given two variables x, z, we translate expressions C in HCL(L) to FO formulas $(\!|C|\!)_{x,z}$ such that $(u,u') \in [\![C]\!]^{t,\alpha}$ iff $t, \alpha[x \mapsto u, z \mapsto u'] \models (\!|C|\!)_{x,z}$

$$
\begin{array}{ll}
(\!|b|\!)_{x,z} = b(x,z) & \\
(\!|C/C'|\!)_{x,z} = (\!|C|\!)_{x,y} \wedge (\!|C'|\!)_{y,z} & y \text{ fresh} \\
(\!|y|\!)_{x,z} = x{=}y \wedge y{=}z & \\
(\!|[C]|\!)_{x,z} = (\!|C|\!)_{x,y} \wedge x{=}z & y \text{ fresh} \\
(\!|C \cup C'|\!)_{x,z} = (\!|C|\!)_{x,z} \vee (\!|C'|\!)_{x,z} &
\end{array}
$$

Equality $x{=}y$ can be expressed by $\mathsf{ch}^*(x,y) \wedge \mathsf{ch}^*(y,x)$. For the converse, we translate positive FO formulas ξ back to expressions $(\!|\xi|\!)^{-1}$ in HCL(L), such that $q_{(\!|P|\!)^{-1},\overline{x}}(t) = \{\nu(\overline{x}) \mid t, \nu \models P\}$ for all variables $\overline{x}$ except for those introduced freshly by the back translation.

$$
\begin{array}{l}
(\!|b(x,z)|\!)^{-1} = \mathsf{ch}^*/x/b/z \\
(\!|\xi \wedge \xi'|\!)^{-1} = [(\!|\xi|\!)^{-1}]/[(\!|\xi'|\!)^{-1}] \\
(\!|x{=}z|\!)^{-1} = \mathsf{ch}^*/x/z \\
(\!|\xi \vee \xi'|\!)^{-1} = (\!|\xi|\!)^{-1} \cup (\!|\xi'|\!)^{-1}
\end{array}
$$

□

We next study HCL$^-$ where variable sharing is forbidden. Union-free expressions of this language can be identified with *acyclic conjunctive queries (ACQs)* over signature L when excluding unions.

Let ACQ (L) be the language of ACQs with finite signature L. Expressions of HCL$^-(L)\cap$N$(\cup)$ can be identified with ACQs over L, so that both of them have the same graph.

Query answering for ACQ (L) is the problem of computing $q_{\xi,\overline{x}}(t)$ for a given formula ξ of ACQ (L), a tree t and variables $\overline{x}$. This problem can be reduced to answering ACQs for the relational db which consists of the set of binary relations $db = \{q_b(t) \mid b \in L\}$. The size of this database is $O(|t|^2)$; it can be computed by answering the queries $q_b(t)$ with $b \in L$.

PROPOSITION 7. *Computing answer sets $A = q_{C,\overline{x}}(t)$ for n-ary queries C in* HCL$^-(L)\cap$N$(\cup)$ *is in polynomial time*

$$O(|t^2| \; |C| \; n \; |A| + \sum_{b \in L} p(|b|, |t|))$$

where $p(|b|,|t|)$ is the time for answering the binary query $b(t)$.

This follows from the above reduction to ACQ 's over relational databases and Yannakakis's algorithm [24] that solves ACQ 's for relational databases db in combined linear time $O(|db| \; |Q| \; |Q(db)|)$ where $Q = q_{\xi,\overline{x}}$ for some ξ in ACQ (L) and variables $\overline{x}$.

PROPOSITION 8. *Let L be a language of binary queries that is closed under finite intersections and inverse and which contains ch^*. The following two languages capture the same queries modulo linear time transformations:*

$$\text{ACQ } (L) = \text{HCL}^-(L) \cap \text{N}(\cup)$$

PROOF. We have to show the inclusion from the left to the right. For the other direction, we have already argued that all union-free composition formulas without variable sharing can be expressed by ACQ's over L.

Vice versa, one can back translate those ACQ's which are 1) acyclic digraphs and 2) where no pair of nodes is connected by two edges labeled by different $b \in L$'s. Here one needs that $\mathsf{ch}^* \in L$. Both properties can be enforced under the assumptions of the proposition: to ensure 1) it is sufficient to invert some of the b-labeled edges while relabeling by b^{-1}. For 2), it is sufficient to replace double edges labeled by b and b' by single edges labeled by $b \cap b'$. □

Let ACQ$^\vee$ (L) be the language of finite unions of ACQs over binary queries in L. The many cited Corollary 4.2 of [15] states ACQ$^\vee$ (FO^{bin}) = FO. The following proposition thus proves Theorem 3.

PROPOSITION 9. *The following two languages capture the same queries (modulo at most exponential time):*

$$\text{ACQ}^\vee \text{ (FO}^{bin}\text{)} = \text{HCL}^-\text{(FO}^{bin}\text{)}$$

PROOF. Conversely, every formula of HCL$^-$(L) is equivalent to a finite union over formulas in HCL$^-(L) \cap$ N($\cup$). This can be seen by distributing unions upwards (possibly at the cost of an exponential blowup). Thus, Proposition 8 applied with $L = \text{FO}^{bin}$ yields ACQ$^\vee$ (FO^{bin})= HCL$^-$(FO^{bin}) as required. $\square$

7. N-ARY QUERY ANSWERING

We present a new query answering algorithm for the hybrid composition language HCL$^-(L)$ without variable sharing in compositions. It extends the previous Yannakakis algorithm for ACQs over graphs to unions [24].

Our first objective is to simplify expressions of HCL$^-$(L) so that no unions appear on the left of compositions. The naive idea would be to replace $(C_1 \cup C_2)/C$ by $C_1/C \ \cup \ C_2/C$. Unfortunately, rewriting with this rule may lead to an exponential explosion, since the subformula C is copied. We solve this problem by introducing sharing path expression.

Let $p \in \texttt{Par}$ be a parameter in a fixed set, which refers to expressions. A *sharing expression* D is similar to an expression of HCL(Q) except that it may contain parameters $p \in \texttt{Par}$, while excluding unions as well as parameters on the left of compositions

$$\begin{array}{lcl} E & ::= & x \ \mid \ [D] \ \mid \ b \\ D & ::= & p \ \mid \ D \cup D' \ \mid \ E/D \ \mid \ \texttt{self} \end{array}$$

An equation system $\Delta = [p_1 \mapsto D_1, \ldots, p_n \mapsto D_n]$ is a finite acyclic mapping from parameters to sharing formulas. This means that all p_i are pairwise distinct and that $\texttt{Par}(D_i) \subseteq \{p_{i+1}, \ldots, p_n\}$ for all $1 \leq i \leq n$. Pairs (D, Δ) satisfying $\texttt{Par}(D) \subseteq \texttt{Par}(\Delta)$ define composition formulas D_Δ inductively as follows: $\texttt{self}_\Delta = \texttt{self}, p_\Delta = \Delta(p)_\Delta, x_\Delta = x, b_\Delta = b, [D]_\Delta = [D_\Delta], (E/D)_\Delta = E_\Delta/D_\Delta, (D \cup D')_\Delta = D_\Delta \cup D'_\Delta$. The size of a sharing formula is the number of its nodes. The size of an equation system is the sum of the sizes of its formulas.

LEMMA 3. *Every composition formula C can be transformed in linear time to a pair (D, Δ) so that $C = D_\Delta$. The sizes of D and Δ are linear in the size of C.*

This can be done by applying the following rewrite rule exhaustively, always with fresh p's:

$$(C_1 \cup C_2)/C \ \Rightarrow \ C_1/p \ \cup \ C_2/p \quad \text{where} \quad \Delta(p) = C$$

Once this is done, we will have to replace formulas E on the right of equations by $E/\texttt{self}$. The introduction of self is not essential but will reduce the number of cases in our algorithm.

We present an algorithm computing sets $q_{D_\Delta,\overline{x}}(t)$ such that D_Δ satisfies NVS(/). We start with a filtering algorithm for a fixed tree t and equation system Δ. It computes all truth values $\texttt{MC}(D, u)$ for all $u \in \texttt{nodes}(t)$ and an input formula D:

$$\texttt{MC}(D, u) = 1 \text{ iff } \exists \alpha. \exists u' \in \texttt{nodes}(t). (u, u') \in [\![D_\Delta]\!]^{t,\alpha}$$

By using dynamic programming, we will compute the values $\texttt{MC}(D, u)$ at most once for all subformulas of D and Δ. There is only a linear number of them by Lemma 3.

$$\begin{array}{lcl} \texttt{MC}(\texttt{self}, u) & = & 1 \\ \texttt{MC}(b/D, u) & = & \bigvee_{(u,u') \in q_b(t)} \texttt{MC}(D, u') \\ \texttt{MC}(p, u) & = & \texttt{MC}(\Delta(p), u) \\ \texttt{MC}([D]/D', u) & = & \texttt{MC}(D, u) \wedge \texttt{MC}(D', u) \\ \texttt{MC}(x/D, u) & = & \texttt{MC}(D, u) \\ \texttt{MC}(D \cup D', u) & = & \texttt{MC}(D, u) \vee \texttt{MC}(D', u) \end{array}$$

The rules for $\texttt{MC}(x/D, u)$ are correct since $x \notin \texttt{Var}(D_\Delta)$ by NVS(/) so that $\texttt{MC}(D, u)$ can be satisfied independently of the value of x. An analogous argument applies to $\texttt{MC}([D]/D', u)$. We let $L(C)$ be the set of binary queries $b \in L$ occurring in C, for an HCL(L)-formula C.

PROPOSITION 10. *Let D be a sharing formula, Δ be an equation system, and t be a tree. Computing table MC on inputs D, Δ and t is in time*

$$O(\sum_{b \in L(D_\Delta)} p(|b|, |t|) + |t|^2(|D| + |\Delta|))$$

if query answering for L is in time $p(|b|, |t|)$ for some function p.

PROOF. For $u \in \texttt{nodes}(t)$, and $b \in L$, let $S_{u,b} =_{def} \{u' \mid (u, u') \in q_b(t)\}$. As a first step, all binary queries occurring in D_Δ are precompiled in a data structure that returns in time $|S_{u,b}|$ the set $S_{u,b}$, for any $u \in \texttt{nodes}(t)$ and $b \in L$. This can be done in time $O(\sum_{b \in L(D_\Delta)} p(|b|, |t|))$. Computing MC with memoization can be done in time $O(|t|^2|D_\Delta|)$, i.e. $O(|t|^2(|D| + |\Delta|))$. $\square$

The query answering algorithm in Fig. 8 processes the input formula D recursively, while filtering unsatisfiable cases (in constant time by using the table MC), eliminating duplicates, and memoizing auxiliary results $\texttt{vals}(D_0, u)$ by dynamic programming so that they are never recomputed.

A partial valuation is a function of type $V \rightarrow \texttt{nodes}(t)$ for some subset $V \subseteq \{\overline{x}\}$; all other variables are projected away. Let ϵ be the partial valuation with empty domain, and $\alpha[x \mapsto u]$ the extension of α mapping x to u. If α and α' are valuations with different domains, then we write $\alpha \cdot \alpha'$ for disjoint union of both functions. Variables of $\overline{x}$ but not occurring in D_Δ can bind to arbitrary nodes of t. Function $\mathsf{extend}_{t,X}$ takes care of this. It extends a partial valuation in all possible ways, so that it becomes total on a finite set of variables X.

The algorithm stores sets of partial valuations $\texttt{vals}(D_0, u)$ as auxiliary results. All these partial valuations can be extended to some whole solution, since unsatisfiable cases are filtered by using MC table.

PROPOSITION 11. *The algorithm in Fig. 8 computes answer set of n-ary queries $A = q_{D_\Delta,\overline{x}}(t)$ in time $O((|D|+|\Delta|)\,|t|^2 n\,|A|)$, if D_Δ belongs to* HCL$^-(L)$.

PROOF. Since every recursive call to vals filters all unsatisfiable cases, every intermediate result can be extended to a whole solution. Duplicates generated by union and projection are eliminated, so that every recursive call returns at most $|A|$ tuples. The worst case time complexity occurs for the b/D' case, and is in $O(|t| + n\ |t|\ |A|)$. Indeed, there are at most $|t|$ successors u by b, hence they are returned in $|t|$ steps at most (since every binary query has been first precomputed in a data structure which returns all the successors of a node, by a particular binary query, in linear time). Hence, the algorithm performs at most $|t|$ unions of sets of size at most $n|A|$, which can be done in time $O(|t|\ n\ |A|)$.

Finally, since we use memoization, intermediate results are never recomputed, so that there are at most $O(|t|\ (|D + \Delta|))$ recursive calls to vals which costs at most $O(|t|\ n\ |A|)$ steps. Hence, line 15 requires at most $O((|D| + |\Delta|)\ n|t|^2|A| + |t||A|) = O((|D| + |\Delta|)\ n|t|^2|A|)$ steps. $\square$

COROLLARY 3. *Answer sets of n-ary queries $A = q_{C,\overline{x}}(t)$ defined in* HCL$^-(L)$ *can be computed in time*

$$O(\sum_{b \in L(C)} p(|b|, |t|) + n|C||t|^2|A|)$$

if query answering for L is in time $p(|b|, |t|)$ for some function p.

```
1  q(D, Δ, x̄, t) =
2   let vals(D_0, u) =
3                 if MC(D_0, u)
4                 then case D_0
5                       of self return {ε}
6                       of p return vals(Δ(p), u)
7                       of b/D' return ⋃_{(u,u')∈q_b(t)} {α | α ∈ vals(D', u')}
8                       of x/D' if x ∈ x̄
9                               then return {α[x ↦ u] | α ∈ vals(D', u)}
10                              else return vals(D', u)
11                      of [D']/D'' return {α' · α'' | α' ∈ vals(D', u), α'' ∈ vals(D'', u)}
12                      of D' ∪ D'' return extend_{t,Var((D'∪D'')_Δ)∩x̄}(vals(D', u))
13                                        ∪ extend_{t,Var((D'∪D'')_Δ)∩x̄} vals(D'', u)
14                else return {}
15        partial_vals=⋃_{u∈nodes(t)} vals(D, u)
16        valuations=extend_{t,x̄}(partial_vals)
17    in
18      return {α(x̄) | α ∈ valuations}
```

Figure 8: Compute answer set $q_{D_\Delta,\overline{x}}(t)$ with implicit memoization.

PROOF. It is an obvious corollary of Lemma 3, and Propositions 10 and 11. □

8. PROVING FO COMPLETENESS

For sake of self containedness, we present an alternative proof of Theorem 3 stating $\text{HCL}^-(\text{FO}^{bin}) = \text{FO}$. It strengthens a result of Schwentick [19, 18] relying on Ehrenfeucht-Fraïssé games and Shelah's decomposition method [20, 23, 17]. Basically, a decomposition theorem relates a logic on compound structures to the logics on its components.

We treat the case of binary trees in a first step, and then lift the result to unranked trees via binary encoding. We consider the FO logic over binary trees, whose signature is $\{(\mathsf{lab}_a)_{a\in\Sigma}, \mathrm{ch}_1, \mathrm{ch}_2, \mathsf{ch}^*\}$. Here, we write ch_1 for the first-child relation and ch_2 for the second child relation. In this section, FO stands for the set of formulas over this signature.

We first recall some definitions from finite model theory (see [11] for more details). Let $n \in \mathbb{N}$ be a natural number and t a tree. For a tuple of nodes $v_1, \ldots, v_k \in \mathtt{nodes}(t)$, we denote by $(t, v_1, \ldots, v_k)$ the extension of structure t by k distinguished nodes $v_1, \ldots, v_k$ (i.e. structure t whose signature is extended by k constant symbols interpreted by v_i's).

The term $\text{type}_n(t, v_1, \ldots, v_k)$ stands for a set of FO formulas $\phi(x_1, \ldots, x_k)$ of quantifier depth at most n that are satisfied when mapping x_i to v_i for all $1 \leq i \leq n$:

$$t \models \phi(v_1, \ldots, v_k)$$

Given two extended structures $(t, \overline{v})$ and $(t', \overline{v'})$, we write $(t, \overline{v}) \equiv_n (t', \overline{v'})$ iff $\text{type}_n(t, \overline{v}) = \text{type}_n(t', \overline{v'})$. It is well-known that formula $\text{type}_n(t, v_1, \ldots, v_k)$ can be characterized by *n-Hintikka formulas*. These are formulas $\tau_{t,v_1,\ldots,v_k}(x_1, \ldots, x_k)$ of FO with k free variables and quantifier-depth at most n that satisfy

$$t' \models \tau_{t,v_1,\ldots,v_k}(v'_1, \ldots, v'_k) \text{ iff } (t, v_1, \ldots, v_k) \equiv_n (t', v'_1, \ldots, v'_k)$$

Finally, every FO formula of quantifier-depth at most n is equivalent to a finite disjunction of n-Hintikka formulas. The set of all n-Hintikka formulas is finite modulo logical equivalence.

We now state our decomposition lemma for binary trees. Note that this lemma is more general than Proposition 4.8 of Neven and Schwentick [18], which is restricted to a single distinguished node, i.e., to the case $n = 1$. Compared to Schwentick's Lemma 4.1 of [19], on one side, our lemma is simpler since it is stated in binary trees, and on the other side, it becomes more easy to explicit non-variable sharing when translating an FO formula into an equivalent $\text{HCL}(\text{FO}^{bin})$ formula.

Given a tree t and a node $u \in \mathtt{nodes}(t)$, we write $t|_u$ for the subtree of t rooted by u.

LEMMA 4 (DECOMPOSITION LEMMA). *Let $m \geq 2$ and $n \geq 0$. Let t and t' be binary trees. Let $\overline{v} = (v_1, \ldots, v_m)$ be an m-tuple of nodes in t and $\overline{u} = (u_1, \ldots, u_m)$ an m-tuples of nodes in t'. We assume that both of these sequences contain at least two different nodes. Let v_a be the least common ancestor of $\overline{v}$ and u_a the least common ancestor of $\overline{u}$. Let v_a^1 and u_a^1 be the first children of respectively v_a resp. u_a, and v_a^2 and u_a^2 their second children. We define set E, L, R of nodes as follows:*

$$E = \{e \mid v_a = v_e\} \qquad L = \{l \mid v_a^1 \lhd^* v_l\} \qquad R = \{r \mid v_a^2 \lhd^* v_r\}$$

In this situation, $(t, v_1, \ldots, v_m) \equiv_n (t', u_1, \ldots, u_m)$ holds under the following three hypothesis

$$\begin{array}{lll} (t, v_a, (v_e)_{e\in E}) & \equiv_n & (t', u_a, (u_e)_{e\in E}) \\ (t_{v_a^1}, (v_l)_{l\in L}) & \equiv_n & (t'_{u_a^1}, (u_l)_{l\in L}) \\ (t_{v_a^2}, (v_r)_{r\in R}) & \equiv_n & (t'_{u_a^2}, (u_r)_{r\in R}) \end{array}$$

PROOF. The proof is based on Ehrenfeucht-Fraïssé (EF) games, which permit to characterize $\equiv_n$ equivalence of two extended structures. The idea is to combine the winning strategies of the Duplicator on the parts of the composition to define a winning strategy of the Duplicator for the whole structures. □

Proving Theorem 3 for binary trees. Let $\phi(\overline{x})$ be an FO-formula with m free node variables $\overline{x}$. We will construct a formula $C(\overline{x}) \in \text{HCL}(\text{FO}^{bin})$ defining the same query as ϕ. The proof goes by induction on m. The case $m = 0$ and $m = 1$ are easy.

The induction step construction mimics the decomposition lemma, in such a way that first, an FO^{bin}-formula selects a branching node v_a, intended to be the least common ancestor of the nodes selected below it. A disjunction enumerates all the subset V of variables of $\overline{x}$ depending on whether v_a is captured by variables from V. Then another disjunction enumerates all the possible 2-partitions $V_1 \oplus V_2$ of the remaining set of variables, and and the construction is applied inductively on the first and second child of v_a respectively, with remaining sets V_1 and V_2 respectively (in the following

proof, the case $V_1 = V_2 = \emptyset$ is considered as a particular case). At each step, the set of variables decreases, hence the construction terminates. We now give the construction formally.

For more convenience, let us write $/\!\!/_{i=1}^{p} C_i$ for $C_1/\ldots/C_p$, and we say that an FO-formula $\gamma(x,\overline{x})$ is *subtree-restricted wrt* x if variables from $\overline{x}$ and quantified variables denotes nodes below x [19].

Given a tuple $\overline{v} = (v_i)_{i\in I}$, and $J \subseteq I$, we denote by $\Pi_J(\overline{v})$ the tuple $(v_j)_{j\in J}$, and we write $\Pi_i(\overline{v})$ instead of $\Pi_{\{i\}}(\overline{v})$. We denote by $\mathrm{ch}_1(v)$ (resp. $\mathrm{ch}_2(v)$) the first (resp. the second) child of $v \in \mathtt{nodes}(t)$, and by $\mathtt{lca}(v,v')$ the least common ancestor of v and v'. Let n be a natural such that $\mathtt{qr}(\phi) \leq n$. For any k in $\{0,\ldots,m\}$, we define $\tau_1^k(y_1,\ldots,y_k),\ldots,\tau_{p_k}^k(y_1,\ldots,y_k)$ as an enumeration of the n-Hintikka formulas with k-free variables (enumeration exists, see [11]). By induction hypothesis, for $k < m$, we can associate with each n-Hintikka formula τ_j^k an equivalent HCL(FO^{bin})-formula C_j^k. Moreover, there exists a finite set of indexes I such that $\phi(x_1,\ldots,x_m) = \bigvee_{i\in I} \tau_i^m(x_1,\ldots,x_m)$ [11]. Now we consider two cases depending on whether all free variables will be instantiated by the same node or not. In the first case, it is obvious to define an HCL(FO^{bin}) formula $\gamma_{\mathtt{eq}}$ equivalent to ϕ. We take $\gamma_{\mathtt{eq}}(x_1,\ldots,x_n) = c/\ /\!\!/_i\ x_i$, where c is the formula $\phi(x,y)$, subtree restricted wrt x, and in which all the x_i's have been replaced by y. The second case is more technical. For $i \in I$, we define D_i the finite set of all tuples (P_E,P_L,P_R,j,k,l) such that

- $\{P_E,P_L,P_R\}$ is a partition of $\{1,\ldots,m\}$ with at most one empty set and
- there exists $t,\overline{v}$ such that $t \models \tau_i^m(\overline{v})$ and
  - $t \models \tau_j^{|P_E|+1}(\mathtt{lca}(\overline{v}),\Pi_{P_E}(\overline{v}))$ and $\forall e \in P_E, \Pi_e(\overline{v}) = \mathtt{lca}(\overline{v})$
  - $t|_{\mathrm{ch}_1(\mathtt{lca}(\overline{v}))} \models \tau_k^{|P_L|}(\Pi_{P_L}(\overline{v}))$
  - $t|_{\mathrm{ch}_2(\mathtt{lca}(\overline{v}))} \models \tau_l^{|P_R|}(\Pi_{P_R}(\overline{v}))$

We let

$$\gamma_{\mathtt{neq}}(x_1,\ldots,x_m) = \cup_{i\in I} \cup_{(P_E,P_L,P_R,j,k,l)\in D_i} C_{(P_E,P_L,P_R,j,k,l)}$$

where the formula $C_{(P_E,P_L,P_R,j,k,l)}$ is equal to

$$c_a/\ /\!\!/_{p\in P_E}\ x_p/\ [c_1/C_k^{|P_L|}(\Pi_{P_L}(\overline{x}))]\ /\ [c_2/C_l^{|P_R|}(\Pi_{P_R}(\overline{x}))]$$

and where (i) c_a is the formula $C_a(x,y)$, being equal to the n-Hintikka formula $\tau_j^{|P_E|+1}(y_1,\ldots,y_{|P_E|+1})$ subtree restricted wrt x, and in which all y_i's have been replaced by y, and (ii) c_ℓ is the formula $\mathsf{child}_\ell(z_1,z_2)$ for ℓ in $\{1,2\}$. Note that composition formulas $C_k^{|P_L|}$ and $C_l^{|P_R|}$ are well-defined, since $|P_L|,|P_R| < m$. Finally we take $C = \gamma_{\mathtt{eq}} \cup \gamma_{\mathtt{neq}}$.

It remains to prove that $t,\overline{v} \models \phi(\overline{x})$ iff $t,\overline{v} \models C(\overline{x})$. When all elements of $\overline{v}$ are equal, it is straightforward to prove that $t \models \gamma_{\mathtt{eq}}(\overline{v})$ in both directions. Suppose now that $\overline{v}$ contains at least two different nodes. We prove the following to be equivalent:

(1) $t \models \phi(\overline{v})$

(2) $\exists i \in I,\ t \models \tau_i^m(\overline{v})$

(3) $\exists i \in I, (P_E,P_L,P_R,j,k,l) \in D_i$ s.t.

(a) $t \models \tau_j^{|P_E|+1}(\mathtt{lca}(\overline{v}),\Pi_{P_E}(\overline{v}))$ and $\forall e \in P_E, \Pi_e(\overline{v}) = \mathtt{lca}(\overline{v})$

(b) $t|_{\mathrm{ch}_1(\mathtt{lca}(\overline{v}))} \models \tau_k^{|P_L|}(\Pi_{P_L}(\overline{v}))$

(c) $t|_{\mathrm{ch}_2(\mathtt{lca}(\overline{v}))} \models \tau_l^{|P_R|}(\Pi_{P_R}(\overline{v}))\}$

(4) $t,\overline{v} \models \gamma_{\mathtt{neq}}$

We only give the proof of (2) $\leftrightarrow$ (3) since other equivalences are easy. Suppose that (2) holds for some i (assume to be fixed from now on), and let P_E be the set of indexes of elements of $\overline{v}$ equal to $\mathtt{lca}(\overline{v})$, let P_L(resp. P_R) be the set of indexes of elements of $\overline{v}$ below the first (resp. the second) child of $\mathtt{lca}(\overline{v})$ (note that these two children exist as there exists two different elements in $\overline{v}$ and all inner nodes have two children). By definition of n-Hintikka formula, there exists (j,k,l) such that (a), (b) and (c) hold. By hypothesis, one gets $(P_E,P_L,P_R,j,k,l) \in D_i$.

Conversely, suppose (3): the set D_i is nonempty. Thus there exists a tree t' and a tuple of nodes $\overline{v'}$ such that the conditions in the definition of D_i hold for t' and $\overline{v'}$. Combining it with the hypothesis, one gets the conditions of the decomposition lemma – for instance, $t \models \tau_k^{|P_L|}(\Pi_{P_L}(\overline{v}))$ and $t' \models \tau_k^{|P_L|}(\Pi_{P_L}(\overline{v'}))$, so that $(t,\Pi_{P_L}(\overline{v})) \equiv_n (t',\Pi_{P_L}(\overline{v'}))$. Thus $(t,\overline{v}) \equiv_n (t',\overline{v'})$, and $t \models \tau_i^m(\overline{v})$.

Concerning the construction of formula C, remark that by definition, $\Pi_{P_E}(\overline{x})$, $\Pi_{P_L}(\overline{x})$ and $\Pi_{P_R}(\overline{x})$ are pairwise disjoint, so that we ensure C to satisfy NVS($/$).

From binary to unranked trees. We use the binary encoding *firstchild-nextsibling* of unranked trees into binary trees [14]. For a binary query $b \in L$, over binary encodings of unranked trees, we denote by $bin^{-1}(b)$ the equivalent query (if it exists) on corresponding unranked trees. It is not difficult to see that a composition of queries b over binary encodings of unranked trees is equivalent to the same composition of queries $bin^{-1}(b)$ over unranked trees. For instance, the query $b/b'/x$ on binary encodings of unranked trees is equivalent to the query $bin^{-1}(b)/bin^{-1}(b')/x$ over unranked trees. Note that it preserves non-variable sharing. It remains to prove that every FO formula over unranked trees is equivalent to some FO-formula over binary trees, modulo *firstchild-nextsibling* encodings, and conversely. This is technical but not difficult.

9. RELATED WORK

The idea of combining Boolean, unary, or binary queries into n-ary queries is not new.

Schwentick [19] shows how to define n-ary queries by combining unary queries by means of regular path expressions of unary formulas of several fragments of FO. He states a decomposition lemma similar to Lemma 4, from which he proves that combination of unary FO queries is complete for n-ary FO (Proposition 4.2). We can directly use this proposition to prove that HCL(FO^{bin}) = FO. What remains unclear, however, is whether the formula obtained does belong to HCL$^-$(FO^{bin}). This is since no explicit construction is presented. Is is not unlikely that the decomposition Lemma 4.1 of [19] leads to a variable sharing, since the decomposition does not partition the tree. The same problem in the case of binary trees is solved by Marx [15] in his proof of Corollary 4.2. We lift this result to unranked trees by binary encoding, while preserving non-variable sharing.

Arenas, Barcelo and Libkin [2] present a similar composition language to HCL(L). While we combine binary queries, [2] provides a mechanism to combine Boolean and unary queries. They give sufficient expressiveness conditions on the basic Boolean and unary query languages to get an FO (resp. MSO)-complete query language (Theorem 2). In Theorem 3, they relate the model-checking complexity of the combined language to the model-checking complexities of the basic query languages.

In the present paper, we focus on query answering for n-ary queries, which was left open by [2]. They investigate model-checking only, where non-variable sharing does not matter. Although the result $\mathrm{FO} \subseteq \mathrm{HCL}(\mathrm{FO}^{bin})$ is an obvious corollary of Proposition 1 of [2] (as soon as we can express the least common ancestor), it is not clear whether it proves that $\mathrm{FO} \subseteq \mathrm{HCL}^{-}(\mathrm{FO}^{bin})$. In particular, it is not clear whether some fragment of the combination language of [2] does correspond to our fragment without variable sharing $\mathrm{HCL}^{-}(\mathrm{FO}^{bin})$. It may be definable by introducing a notion of acyclicity as in conjunctive queries, but it not as simple as our syntactic restrictions.

10. CONCLUSION AND FUTURE WORK

We have distinguished a polynomial time fragment of *Core XPath 2.0*, the polynomial time path language PPL. We have shown that PPL is FO complete. No fragment of FO logic in ordered trees is known that enjoy these two properties simultaneously.

This shows that the syntax of FO logics is too poor to talk about trees simultaneouly in an efficient and FO-expressive manner. It justifies path expressions as foundation of XPath a posteriori from a fundamental perspective. An alternative solution would have been to use modal logic operators in combination with variables such as in hybrid logic [2, 6].

The first problem of FO in ordered trees is that it does not enjoy quantifier elimination, while path languages with variables like *Core XPath 2.0* do. The missing expressiveness of FO is remedied by allowing for path composition.

The second problem of FO logic is that it is difficult to distinguish polynomial time fragments beyond acyclic conjunctive queries. As we have shown in this paper, this limitation can be overcome in a path based language with variables such as *Core XPath 2.0*. The most important restriction is to forbid variable sharing in path compositions.

A number of open issues remain. First of all, the new query algorithm for PPL presented here requires cubic time in the size of the tree. The question is, whether larger linear time fragments of PPL can be distinguished that go beyond the variable free fragments such as *Core XPath 1.0*.

The close correspondence between acyclic conjunctive queries [9] and the hybrid composition language without variable sharing in path composition – that we have presented – opens new questions about enumeration algorithms [3, 8, 10]. Which fragments of ACQs or HCL admit polynomial-time preprocessing and a linear enumeration delay?

Finally, let us note that a very similar proof as presented here can be used to show that $\mathrm{HCL}^{-}(\mathrm{MSO}^{bin}) = \mathrm{MSO}$.

Acknowledgments It's a pleasure to thank Arnaud Durand for many interesting suggestions on earlier versions of the paper. Thanks to Balder ten Cate for discussions and deep remarks. We are also grateful to Luc Ségoufin for pointing out the composition method, and to anonymous referees for their valuable comments.

11. REFERENCES

[1] N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13(6):939 – 956, 2003.

[2] Marcelo Arenas, Pablo Barcelo, and Leonid Libkin. Combining temporal logics for querying XML documents. In *International Conference on Database Theory*, 2007.

[3] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL*, 2006.

[4] Pablo Barcelo and Leonid Libkin. Temporal logics over unranked trees. In *LICS*, pages 31–40. 2005.

[5] Michael Benedikt and Christoph Koch. XPath leashed. 2006. submitted.

[6] Patrick Blackburn and Jerry Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4(3):251–272, 1995.

[7] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *ACM conference on Theory of computing*, 1987.

[8] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. 2006. submitted.

[9] Arnaud Durand and Etienne Grandjean. The Complexity of Acyclic Conjunctive Queries Revisited. 2004.

[10] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logics*, 2006.

[11] H. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer Verlag, Berlin, 1999.

[12] M. Franceschet and E. Zimuel. Modal logic and navigational xpath: an experimental comparison. In *Proceedings of Workshop Methods for Modalities*, pages 156–172, 2005.

[13] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

[14] Leonid Libkin. Logics over unranked trees: an overview. *Logical Methods in Computer Science*, 3(2):1–31, 2006.

[15] Maarten Marx. Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959, 2005.

[16] Maarten Marx. First order paths in ordered trees. In *International Conference on Database Theory*, pages 114–128, 2005.

[17] F. Moller and A. Rabinovich. Counting on CTL: on the expressive power of monadic path logic. *Information and Computation*, 184(1):147–159, 2003.

[18] Frank Neven and Thomas Schwentick. Query automata over finite trees. *TCS*, 275(1-2):633–674, 2002.

[19] Thomas Schwentick. On diving in trees. In *MFCS*, pages 660–669, 2000.

[20] S. Shelah. The monadic theory of order. *Annals of Mathematics*, 102:379–419, 1975.

[21] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory*. PhD thesis, Department of Electrical Engineering, MIT, 1974.

[22] Balder ten Cate and Maarten Marx. Axiomatizing the logical core of XPath 2.0. In *ICDT*, 2007.

[23] Wolfgang Thomas. Logical aspects in the study of tree languages. In *9th International Colloquium on Trees in Algebra and Programming*, pages 31 – 50, 1984.

[24] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceeding of VLDB*, pages 82–94, 1981.